P C VEKATS

1997

ANDRES A. NAVARRO N.
The Implementation of a Windows 95 based Virtual Environments Knee Arthroscopy Training System (PC VEKATS)

being a dissertation submitted in partial fulfilment of the requirements for the Degree of Master of Science

in the University of Hull

by

Andrés Adolfo NAVARRO NEWBALL

September 1997
... To my parents

Acknowledgements

Many thanks to all the lecturers of this MSc course for the knowledge imparted and a very special thank to this projects' supervisor Mr. Derek Wills and to Mr. Ivan Logan for their time and patience during the development of PC VEKATS.
Figures

FIGURE 2.1. TYPICAL ARTHROSCOPY, 4
FIGURE 2.2. THE ANATOMY OF THE KNEE, 5
FIGURE 2.3. ARTHROSCOPE AND PROBE, 6
FIGURE 2.4. MARCHING CUBE, 8
FIGURE 2.5. DELAUNAY TRIANGULATION, 10
FIGURE 2.6. VOXMAP AND POINTSHELL 12
FIGURE 2.7. POINTSHELL CREATION FROM DECIMATED VOXELISED SURFACE, 13
FIGURE 2.8. SINGLE SPRING-AND-DASHPOT CONNECTION, 15
FIGURE 2.9. HIERARCHIES FOR THE KNEE, 19
FIGURE 3.1. PROJECT PLAN 32
FIGURE 4.1. CLASSES THAT BUILD UP THE INTERFACE 35
FIGURE 4.2. CLASSES THAT BUILD UP THE ARTHROSCOPIC SIMULATION 38
FIGURE 4.3. CLASSES THAT BUILD UP THE FINITE ELEMENT MODELLING OBJECTS 40
FIGURE 4.4. THE VEKATS SYSTEM, 42
FIGURE 4.5. VEKATS' INTERFACE, 43
FIGURE 4.6. THE PC VEKATS' INTERFACE 45
FIGURE 4.7. OPEN SIMULATION 46
FIGURE 4.8. CONTROL PANEL 47
FIGURE 4.9. INSTRUMENT PANEL 49
FIGURE 4.10. VIDEO WINDOW 49
FIGURE 6.1. PC VEKATS' PERFORMANCE 67
Tables

TABLE 6.1. COMPRESSION TIMES AND FILE SIZES (AVI), 62
TABLE 6.2. COMPRESSION TIMES AND FILE SIZES, 62
TABLE 6.3. FILE SIZE VS. FRAME SIZE, 63
TABLE 6.4. FILE SIZE VS. FRAME RATE, 63
TABLE 6.5. MPEG FORMAT, 64
TABLE 6.6 PC VEKATS' PERFORMANCE 66
Chapter 1: Introduction

Knee arthroscopy (see section 2.1) is a form of Minimal Access Surgery (MAS) used to examine, diagnose and treat problems in the knee joint. In this kind of surgery a small incision is made, minimising the disruption to surrounding tissue. In order to perform such an operation a specialised set of tools is required (e.g. arthroscope, probe, grasper and scissors). Also, an arthroscopic surgeon must acquire different skills including triangulation, orientation and dexterity, since the manipulation of the instruments can cause confusion. It is vital to reconcile the visual cues with the kinaesthetic cues, because a misuse of the tools can cause serious problems in the patient (e.g. degenerative arthritis).

One of the problems of arthroscopic surgery is that there is a long learning curve associated with MAS. There is no standard training system in knee arthroscopy. However, some training can be done away from the operating theatre. As a start, the trainee can operate within a box structure in order to gain the basic orientation and dexterity skills (e.g. peeling grapes with a blade tool). Later, operative procedures may be simulated by means of artificial knees, which are inaccurate, expensive to maintain and have limitations like the impossibility of working in a ‘wet’ environment. A further stage, could be the use of pigs’ knees, which are anatomically similar to humans’ knees and allow practice in a cheap, organic environment. Nevertheless, this model could cause cultural aversions. After having passed the stages described above, the trainee can participate in supervised arthroscopic procedures on humans. Basically, the process consists on learning the procedure, observing a skilled surgeon, performing a supervised procedure and teaching to the next intake of trainees. In a training program like this, assessment data is difficult to obtain at each stage thus there is a need for an objective way to monitor the progress of the trainees.

The aim of the Virtual Environments Knee Arthroscopy Training System (VEKATS) project is to supplement the current training procedures available to orthopaedic surgeons by means of a cheap non-destructive environment where not only the basic skills required are gained but also precise assessment data can be manipulated and fed back. Currently non-diagnostic procedures are beyond its current scope. In order to
accomplish this, researchers of the Computer Science Department at the University of Hull have a number of projects that together will constitute the required system. This MSc dissertation, PC (personal Window 95 based computer) VEKATS, is one such project.

PC VEKATS is a computer simulation of a knee arthroscopic diagnostic procedure that will be used for training purposes. Its aim is to implement a system like VEKATS in a Microsoft Windows environment. The author used, improved, adapted and changed when necessary the code obtained from the original implementation of VEKATS.

This report is broken down into the following chapters: Chapter 2 explains the theory used to develop PC VEKATS. It includes an overview of the arthroscopic procedure, followed by an explanation of the techniques used in the project (e.g. representation of objects, collision detection techniques, etc). Next, a description of the different video playback formats available for the PC is given. The chapter ends with a description of VEKATS and its current status. Much of the information in this chapter was developed from Logan (1997).

Chapter 3 explains the objectives and the scope of the project and shows the plan that was adopted in order to develop it.

Chapter 4 explains the design of the project. The changes that this design had over the design of VEKATS are also highlighted. Finally, the design of the interface and the guidelines and techniques that were taken into account in order to develop it are explained, again contrasting with VEKATS.

Chapter 5 describes the implementation of the project. First, the base code used is described briefly. This is followed by a comparison of the implementation alternatives in terms of interactivity, interface and video playback, explaining the choice made for PC VEKATS. Next, the changes that were made to the implementation of VEKATS are explained. Finally, the magnetic input device is introduced. A discussion of the problems encountered in the implementation process is included.
Chapter 6 shows the results obtained in terms of performance and usability. It includes the evaluation of the performance of PC VEKATS in different situations (e.g. switching on/off the anatomic overview, colliding or not, switching on/off the video clip, etc). A comparison of the different video formats is also shown.

Finally, Chapter 7 ends the document with a set of conclusions and recommendations for future work.
Chapter 2: Background

This chapter describes the theory used to develop PC VEKATS. It starts with a brief description of knee arthroscopy and its tools. This is followed by an explanation of a number of simulation techniques (e.g. representation of objects, collision detection, etc). Next, the video formats available for PC are discussed. The chapter ends with a description of VEKATS and a set of conclusions.

2.1. Knee Arthroscopy

Knee arthroscopy is a form of Minimal Access Surgery. Figure 2.1 shows the procedure. It starts by administrating an anaesthetic (either local or general). At this stage, small incisions are made on the kneecap and the arthroscope and the tools are inserted. Through a screen, the orthopaedic surgeon looks at the inside of the joint. A sterile fluid may be injected into the joint space to enlarge it and enhance visibility. Finally, the incisions are stitched or in many cases just covered with surgical tape, (Mayo Clinic, 1997).

![Figure 2.1. Typical Arthroscopy,](Applied Medical Informatics, 1996, p.1)
The anatomy of the knee and the tools used in arthroscopy are explained next, since these are the objects to be simulated by PC VEKATS.

Figure 2.2 shows the anatomy of the knee. A hinge joint is formed when the bones of the knee, the femur and the tibia meet. The articular cartilage that covers the ends of the tibia and femur and the underside of the patella helps to cushion this joint. Further cushioning is obtained from pads of cartilage called the meniscus (lateral and medial). Stability in the knee is achieved with the help of the ligaments. The collateral ligaments run along the sides of the knee and limit sideways motion. The anterior cruciate ligament, or ACL, limits rotation and forward motion of the tibia. It connects the tibia to the femur at the centre of the knee. The posterior cruciate ligament, or PCL, limits backward motion of the tibia and is located behind the ACL. (Southern California Orthopaedic Institute, 1997).

![Knee Anatomy Diagram](image)

*Figure 2.2. The Anatomy of the Knee, (Southern California Orthopaedic Institute, 1997, p.1.)*

"An arthroscope is an instrument similar to a telescope. It is approximately as thick as a pencil, and has a lens on the one end. A small video camera is attached to the back of the arthroscope, allowing one to visualise the inside of the joint," (Candor technologies, 1996, p.1). Figure 2.3a shows a diagrammatic representation of an arthroscope. The standard arthroscope used in diagnosis is a 4 to 5 millimetre scope (width of the shaft), with an angle θ of 30° or 70° (direction of view). A greater area of the joint can be
viewed as the scope is turned by angling the direction of the view away from the axis of
the scope. The saline is inserted into the cavity, usually down the arthroscope sheath.
The flow is kept free of floating particles by making a second incision that egress the
saline through an irrigation cannula.

![Diagram of arthroscope and probe](image)

Figure 2.3. Arthroscope and Probe.
Adapted from Logan (1997).

The probe (see figure 2.3b) is a 12 centimetres gadget with a 3.5 millimetre hook of the
tip. It is used to palpate and move the tissues. The probe aids in the diagnosis of
meniscal tears and other pathologies and facilitates a clear field of view for the surgeon.
Other tools like the grasper and the scissors are out of the scope of this discussion, since
the probe is the only one that has been implemented in VEKATS.

### 2.2. Techniques Used

Traditional animation does not require realistic behaviour but behaviour that is
believable. However, when implementing a computer simulation the focus is on trying to
model the physical properties of the objects. In this way all the objects in the scene are
able to behave realistically. Such a behaviour can be simulated through applied physics
(physical based modelling), and the importance of it is the necessity of accuracy,
autonomy and realism in the simulated environment, (Chadwick et. al., 1989). Many
techniques have been implemented in order to simulate a knee arthroscopic diagnostic
procedure in VEKATS. They are described next.
2.2.1. Representation of Objects

The next couple of sections describe the Marching Cubes algorithm and the Delaunay Triangulation method respectively. The first is a common method for meshing objects and the second is a method that helps to automate the mesh generation in deformable objects.

2.2.1.1. The Marching Cubes Algorithm

Medical data (e.g., bones) can be modelled by using the Marching Cubes algorithm. This technique uses a divide-and-conquer approach to generate inter-slice connectivity. The algorithm determines how the surface of the image intersects a cube (voxel), then moves (marches) to the next cube (see figure 2.4). In this way a three dimensional representation of the surface of the object is created, (Lorensen and Cline, 1987).

First, the surface is located in a logical cube created from eight pixels obtained from two adjacent slices, four pixels from each one (see figure 2.4). Then, the surface intersection in a cube is found by assigning a one to a cube’s vertex if the data value at that vertex is greater or equal to the value of the surface that is being constructed. This means that these vertices are inside the surface. On the opposite case, when the values are below the surface (or outside), a value of zero is assigned. Having done this, the topology of the surface within a cube can be determined by finding the location of the intersection of it with the surface. After finding how the surface intersects the current cube, the algorithm moves to the next one.
There are $2^8 = 256$ ways a surface can intersect a cube, because there are eight vertices in each cube and two states (inside or outside). By enumerating these 256 cases a table can be created in order to look up surface-edge intersections, given a label for each vertex in the cube. The table contains the edges intersected for each case. Instead of triangulating the 256 cases, two symmetries of the cube can be used to reduce the problem to 14 patterns. First, rotational symmetry. Second, if the relationship of the surface values to the cubes is reversed the topology of the triangulated surface remains the same. Complementary cases where vertices less than the surface value are interchanged with those with greater value are equivalent. Only cases with zero to four vertices greater than the surface value need to be considered.

Based on the state of each vertex of the cube, an eight bit index can be created for each case. This index contains one bit for each vertex. The index is then used to tell which edge the surface intersects. Having the intersection, it can be interpolated along the edge. The final step of the Marching Cubes algorithm calculates a unit normal for each triangle vertex. This information is used to produce Gouraud-shaded images. In order to get the unit normal at each vertex, the gradient vector at the surface of interest must be estimated first. In order to calculate the gradient at all vertices of the cube, it is
necessary to keep four slices in memory at once. The *Marching Cubes* macro-algorithm is given by Lorensen and Cline (1987, p.166), as follows:

1. Read four slices into memory.
2. Scan two slices and create a cube from four neighbours on one slice and four neighbours on the next slice.
3. Calculate an index for the cube by comparing the eight density values at the cube vertices with the surface constant.
4. Using the index, look up the list of edges from a pre-calculated table.
5. Using the densities at each edge vertex, find the surface-edge intersection via linear interpolation.
6. Calculate a unit normal at each cube vertex using central differences. Interpolate the normal to each triangle vertices and vertex normals.
7. Output the triangle vertices and vertex normals.

2.2.1.2. Delaunay Triangulation

The algorithm found in Cignoni et. al. (1994) is able to produce a three dimensional representation for a deformable object by using the *Delaunay Triangulation* technique. The final result is set of tetrahedral structural elements. This technique is necessary as it is important to automate the generation of the meshes. In the case of VEKATS the algorithm proposed by Cignoni et. al. (1994) is used, despite of the fact it still has some problems. For example, it cannot be applied in regular grid data sets and it may fail when used in large data sets (over twenty thousand points). A different approach to the representation of this soft objects can be found in Wyvill et. al. (1986a).

The *Delaunay Trinagulation* algorithm deals with the vertices, or points in space, of a polygonal model, without taking into account the polygonal information. First, the algorithm creates an edge between a starting point picked from the set and the nearest point to it, (see figure 2.5a). This edge is inserted into an active edge list. The algorithm then starts an iterative process where an edge can either spawn a new triangle, or fail to spawn. If the latter happens, the edge is removed from the active edge list. The loop continues until all edges have been removed from this list. A new triangle is spawned by
an edge when there is a point near to the edge which does not have another point within the circle circumscribed by the vertices of the triangle, (see figure 2.5b). If an edge is taken and it is not possible to construct a triangle which does not contain a fourth point within its boundary circle, it means either that the edge is a boundary edge or that it is grid-locked into triangles around it. In such a situation the edge can be removed from the active edge list, (see figure 2.5c). The Spawnning of a triangle adds new edges to the active edge list so that the triangulation process can continue, (see figure 2.5d).

![Figure 2.5. Delaunay Triangulation. Adapted from Logan (1997).](image)

VEKATS uses low complexity polygonal models in which the Delaunay Triangulation algorithm produces adequate deformable meshes. However, it is known that the Delaunay Triangulation can produce undesirable results. For example, the fact that the algorithm ignores the connectivity from the polygonal database may cause the creation of elements that violate the surface of the object (e.g. the tip and the base of a hook could be linked together). This problem is reduced when the radius of the circle that circumscribes the vertices of the element is reduced. Another problem is that the Delaunay Triangulation may produce highly acute elements thus affecting the work when using the Finite Element Modelling technique (see section 2.2.2.2.3), where it is desirable to have elements with a roughly equilateral aspect. A solution is to detect substantially deformed aspect ratios and re-mesh that area. Nevertheless, this is not a trivial process. An adapted version of the macro-algorithm for the Delaunay Triangulation given by Logan (1997) follows:
with a point p in the set of points P(n)
   select q, nearest point to p
   insert edge p->q in the active edge list
while active edge list not empty
   for each edge a->b
      find nearest point, r, to edge
      if boundary circle of a->b->r empty
         add b->r and r->a edges into the active edge list
         add a->b->r into triangle list
      else
         remove edge a->b from the active edge list

2.2.2. Collision Detection Techniques

This report focus now on the explanation of the collision detection methods used in PC VEKATS. The next few sections describe the Voxel Based Collision Detection technique for rigid objects followed by a discussion of the collision detection techniques available for soft objects (e.g. Spring-and-Dashpot, Reference Frame and Finite Element Modelling).

2.2.2.1. Rigid Objects and the Voxel Based Real Time Collision Detection Method

The Voxel Based Collision Detection method allows a fast and accurate collision detection with rigid objects. The idea of this technique is to discretise and object into a grid of voxels. This is a pre-processing task which results in a Voxmap (see figure 2.6a) where voxels are filled or empty, depending on whether they belong to the object or not. A Pointshell model of the object has to be created as well (see figure 2.6b). In order to detect a collision, the Pointshell of one object is checked against the Voxmap of the other. If a position in the Pointshell corresponds to a filled position in the Voxmap, a collision has occurred. As an extension to the method, a response force can be calculated by assigning penalty forces proportional to the depth of inter-penetration, (Logan and Wills, 1997). Some implementations of this method can be seen in García et. al. (1994) and in VEKATS. The method can only be applied to rigid objects, because a
change of the shape would require a re-calculation of the Voxmap and the Pointshell representation, which in practice cannot be done in real time. The pre-processing steps of the method are explained next.

Figure 2.6. Voxmap and Pointshell

A set of steps required to voxelise an object can be identified. First, the voxel space is allocated depending on the dimension of the object and the desired granularity of the voxel map. Next, a Scan Conversion of each polygon into the voxel space is performed in order to create the voxel shell. The Scan Conversion technique used here is just an extension to three dimensions of the standard two-dimensional Scan Conversion algorithm for pixelisation of polygons. The last step in the voxelisation consists on filling the hollow shell of voxels to produce volume. To accomplish this, a standard seed-fill technique is used to produce a Voxmap which can be associated with the polygonal position and orientation of the object by simple co-ordinate transformations. The seed-fill marks all voxels outside the object. In this representation, the volume of the object is all voxels which have not been marked plus those which make up the surface. This method of filling ensures that objects with open surfaces will be voxelised as such.

There are two ways of encoding the environment as voxels. One is to combine all objects' Voxmaps into one scene Voxmap. This technique is appropriate when many objects are static in the environment, because it frees the computer of detecting potentially statically-interfering objects. However, in a sparse environment most of the Voxmap would be empty. Another technique is to use individual object's Voxmaps, more appropriate for dynamic scenarios like VEKATS, where the knee may be moved by the trainee.
The time to find a collision with the voxel based method is independent of the number, size and shape of the static objects in the scene and from the configuration of the dynamic objects. Accuracy depends on the size of the voxels that discretise the volume of each object. The number of checks made by the collision detection of the algorithm is a linear function of the number of points in the surface of the dynamic object. Nevertheless, it is independent of the number of voxels in the volume of other objects with which the dynamic object can collide. However, if the Voxmap resolution is increased, the performance will be affected since the Pointshell density will also have to be increased in order to avoid stealth voxels from penetrating it. An anti-stealth-voxel algorithm for creating a shell that contains a relatively small number of points is described next, (Logan and Wills, 1997, p.6):

1. Select any voxel in S that has not previously been decimated, and include its centre point P into the output point distribution.
2. Decimate all voxels in S that are wholly enclosed by a sphere of radius D/2 centred at point P.
3. Repeat steps 1 and 2 until all voxels in S have been decimated.

Where D is the size of the stealth box. S is a Voxmap constructed as shown in figure 2.7, using voxels of size L and satisfying D/L > 1

![Figure 2.7. Pointshell Creation From Decimated Voxelised Surface.](image)

(Logan and Wills, 1997, p. 7).
The anti-stealth-voxel algorithm works because if the inscribed sphere of a voxel of size \( D \), which radius is \( D/2 \), cannot penetrate undetected then the voxels itself cannot either. Moreover, if the sphere is to penetrate undetected, its centre point must enter one of the decimated voxels of \( S \). As every point in a decimated voxel lies within a radius \( D/2 \) from the point \( P \) that decimated it, and the inscribed sphere contains \( P \), then the voxel itself cannot penetrate undetected since it contains \( P \). Note that as \( D/L \) increases, the decimating sphere grows to enclose more whole voxels \( N \) and at the same time the Pointshell becomes more optimal, but also requires more pre-processing time. The accuracy increases as the value of \( L \) decreases.

2.2.2.2. Deformable Objects

Much research has been conducted in order to model a collision with a deformable object. For example, a Simulation technique based on a key point representation of the objects is discussed in Wyvill et. al. (1986b). Constrain methods are explained in Isaacs and Cohen (1987), Barzel and Barr (1988), Platt and Barr (1988), Witkin and Kass (1988) and Logan (1997). A more complete model which includes fracture, plasticity and visco-elasticity can be found in Terzopoulos and Fleischer (1988). The models that have been adapted in VEKATS (e.g. Spring-and-Dashpot, Reference Frame and Finite Element Modelling) are explained next.

2.2.2.2.1. The Spring-and-Dashpot Method

In the Spring-and-Dashpot method, an object is represented by a set of mass points connected via a simple spring arrangement to form a mesh, (see figure 2.8). The spring force will tend to restore the mass points to their rest length, a result of applying Hooke's law. The dashpot will behave as a damper that deforms at a rate proportional to the rate of the applied load. This method is easy to understand and to program. However, it is computationally expensive as the numerical effort required increases with the violence of the collision. Further, it can be inaccurate for three dimensional models, (Moore and Wilhelms, 1988).
The Spring-and-Dashpot method is iterative. For each point the acceleration is calculated first from the material properties of the connection which applies force to the point. Having done this, the velocity and then the position of the mass-point at that time step can be obtained. After forces are applied to the object, the method will eventually settle down when the deformed shape is approximated. Equations 2.1, 2.2 and 2.3 model this method.

\[
a_{t+\Delta t} = a_t + \frac{k \cdot dl}{l \cdot m} \quad (2.1)
\]

\[
v_{t+\Delta t} = (v_t \cdot r + a_t) \cdot dt \quad (2.2)
\]

\[
x_{t+\Delta t} = x + v \cdot dt \quad (2.3)
\]

Where:

\[ dt = \text{time step.} \]
\[ a = \text{acceleration.} \]
\[ v = \text{velocity.} \]
\[ l = \text{length of the connection applying the force.} \]
\[ k = \text{spring constant.} \]
\[ r = \text{damping constant.} \]

2.2.2.2.2. The Reference Frame Method

The Reference Frame Method is an extension of the Spring-and-Dashpot method. It increases the rigidity across complex objects. The method avoids the possibility of the
object from collapsing on its own shape. Besides the usual connections between mass points, each mass point is linked to the reference frame of the object, thus separating the deformation of the object from its rigid body motion. The restoring force is proportional to the difference between the current location of the mass point and its reference location. Despite of the fact that this model increases the complexity and the computational cost, it adds usability to the *Spring-and-Dashpot* method. The calculations to get the acceleration have to be obtained from equation 2.4.

\[
a_{t+a} = a_t + \frac{k \cdot dl}{l \cdot m} + \frac{k \cdot d}{m} \quad (2.4)
\]

Where:

d = displacement of the mass point from its reference position, in effect the addition of a connection with initial length of zero.

### 2.2.2.2.3. The Finite Element Modelling Method

The finite element theory seems to be one of the best ways of modelling elastic and plastic deformation of objects, (Gourret et. al., 1989). As Baker and Pepper (1991, p.3) defined it: "*The finite element method is a collection of theory-rich techniques which can produce near-optimal approximate solutions to the initial-boundary value partial differential equations common to engineering and mathematical physics. The implementation ultimately employs subdivision of the problem domain (geometry) into a finite number of small regions called finite elements.*" In the case of computer graphics the object can be decomposed into a mesh of basic elements (e.g. cube, prism, etc.). The idea is to minimise the equilibrium equation across all elements. This technique is the one which is being used in the latest implementation of VEKATS, (Logan and Wills, 1997).

The *Finite Element Modelling* method is a numerical method that requires the discretisation of time and space. In this technique the discretisation starts dividing the region of interest into regions or elements. In order to specify the behaviour of each
element, its displacement fields are defined in terms of its nodal displacements. After the latter is done and knowing which nodes connect specific elements together, a set of system equations can be generated. The set of equations is then solved using boundary conditions, giving the displacement of the nodes in the problem. At this stage, the strains and various stress fields can be calculated. The Lagrangian Mesh Discretisation is usually applied in Finite Element Modelling. A Lagrangian mesh is fixed to the material being modelled and it provides a more accurate description of the boundary of a component, (Boston, 1992).

The elements in the model are linked together by sharing nodes between adjacent elements. The deformation of the object is obtained statically when minimising the equilibrium equation (equation 2.5) across all elements, as follows:

$$W_R = W_B + W_S + W_F \quad (2.5)$$

Where:

$W_R =$ Internal virtual work due to internal stresses.  
$W_B =$ Virtual work due to body forces (e.g. gravity).  
$W_S =$ Virtual work due to surface forces (e.g. pressure).  
$W_F =$ Virtual work done by point/nodal forces.

Shape descriptors and physical attributes determine the geometry and the behaviour of an element, respectively. This data can be used to minimise the energy equation (equation 2.6) by using the matrix form described next, (Gourret et. al., 1989):

$$K \cdot U = R \quad (2.6)$$

Where:

$K =$ Stiffness matrix of the object.  
$U =$ vector that contains all nodal displacements.
R = Vector that with all the applied loads.

As K is known and R can be calculated, U can be found. However computations of K and R are time consuming tasks. The Finite Element Modelling method is a single step process that solves the equation for U when a set of forces R is applied. Thus, the calculations are only required when R is changed and not at every time step.

Tetrahedral elements are more frequently used due to the fact they are usually produced by automatic mesh generation algorithms, like for example the Delaunay Triangulation algorithm, (Sheppard, 1998). Moreover, they are easily tessellated through the shape of the object and, depending on the number of nodes in the edge of each element, the accuracy of the deformation can be increased.

The Finite Element Modelling method is very accurate and does not have the numerical stability problems present in iterative simulation methods (e.g. Spring-and-Dashpot). A more detailed discussion about the method is out of the scope of this work, nevertheless, a full description can be found in Baker and Pepper (1991) or in Fagan (1992).

2.2.2.2.4. Layering for Animation

A hierarchy of composite deformations can be defined and an animation performed. Relationships can be defined among different layers of the model and a set of constrains can be taken into account, (Chadwick et. al., 1989). Figure 2.9a shows how a simple anatomical hierarchy could be the cause of unrealistic movements of the knee. For example, the movement of the tibia would always imply the movement of the ligaments (which are attached to both the tibia and the femur). As a solution to this, and in order to simulate the distraction of the knee (inflation with saline), a hierarchy of systems like the one shown in figure 2.9b can be used. This representation is more accurate. However, this model needs to link positional data between objects like the ligaments and the bones. VEKATS uses this latter approach. Here, a distraction of the knee moves the femur upward. The patella is constrained to the femur and moves with it. The ligaments have to be able to stretch because the tibia and the femur move in opposite directions during the procedure. Conflicting constraints within the ligaments cause a deformation.
Figure 2.9. Hierarchies for the Knee. Adapted from Logan (1997).
2.3. Video Formats

As the video playback system of VEKATS had to be re-implemented in PC VEKATS, a discussion of video formats follows. Much of the information in the following sections was developed from Williams et. al. (1996a), as it was the most up-to-date source of information found by the author of this report. The most common video file formats in use on Windows 95 are MPEG, Quicktime and Video for Windows. Video can add interest, help to explain an abstract concept, show colours, movement and sounds. Thus, some issues to improve its effectiveness should be considered. For example, format, file size, content description, naming conventions and video production considerations, (Internet Video Services, Inc., 1996).

Motion video is made up of a series of still frames (images) played at 25 frames per second (30 in the USA). There are two main areas in digital video. First, animation created entirely in the digital domain (e.g. using software like 3D Studio). Second, video digitised from analogue video sources like the VHS tape, camcoder or live video feed. In education, digital video may provide new possibilities, given that the way to view and interact with it is chosen correctly. For example, information about environments which can be either dangerous or too costly can be provided by means of video (e.g. in medical education real-life situations can be better understood if represented by video). Digital video can also provide a strong motivational incentive to persist in a task. However, if visual elements (e.g. still images, video, etc.) are misused, they can cause misrepresentation and impede learning.

It is important to have awareness of other alternatives which can be more educationally appropriate in various situations. When there are technical reasons why video is not possible, still images can prove a useful alternative to a video sequence. An alternative like animation will only aid learning when changes exist in the direction of the moving object, and not only in its changes over time (motion). The display of an animation simultaneously with an accompanying narration has proven to increase the retention. Imagination can also be stimulated by the creative use of sound. It is important to understand the capabilities provided by each medium, (Williams et. al, 1996a).
2.3.1. Issues and Choices in Digital Video

Digital video files are large. This means that it is not practical to play back a large amount of data per second, even when the storage space is available. Thus, the file size of a video should be reduced to fit the data transfer rate of the final playback system. However, a set of problems associated with digital video can be identified. These are: size of the window, frame rate and quality of image. These problems can be tackled using compression techniques.

2.3.1.1. Size of Video Window

Since a digital video stores a lot of information about each pixel in each frame, the time taken to draw the pixels is less when the window size is small. In contrast, there may not be enough time to display an image (single frame) before it is time to start the next one when the window size is large. Thus, an appropriate window size must be chosen. Full screen video is not usually necessary. Moreover, there are other views that need to be on the screen at the same time. A solution could be to make the size of the window less that the resolution of the original video source (in this case compression techniques have to be used). Other complementary solutions to the problem are the use of a fast hard disk and graphics acceleration hardware.

A more detailed explanation about graphics acceleration hardware is out of the scope of this project since currently there are more than one hundred boards based on more than twenty 3D chipsets. However, it is important to note that some of these boards have been built to run on different platforms (e.g. Windows 95, Mac, etc) and many of them support 3D libraries like OpenGL. A better discussion on this topic can be found in Fourth Dimension (1997). A comparison of the performance of PC VEKATS using two of the graphic cards available in the department of Computer Science at the University of Hull (e.g. Matrox millenium, STB Velocity) is given in section 6.2.
2.3.1.2. Frame Rates

The time available to display a frames’ pixels may not be enough. Some solutions exist to this problem. For example, compression of data reduces the amount of information that needs to be transferred from disc to screen. Additionally, with smaller window sizes, video played at reduced frame rates is acceptable.

2.3.1.3. Image Quality

The quality of the original source and the degree of compression used affects the quality of the image. The relationship between file sizes and quality varies considerably for different Compressors/Decompressors (CODECS). The concept of CODEC is explained in the next section.

2.3.2. Compression and Decompression

Compression is achieved using algorithms which identify the information that needs to be recorded and stored. This information is then reconstructed during decompression. A CODEC compresses the video in the digitising process and decompresses it during playback. Software CODECS are usually optimised for decompression and are required for the playback of the video in the user’s computer. They are also easily upgraded by installing new versions of the driver.

Two types of compression exist. In Lossless compression, all the data is preserved and typically will compress images to a ratio of 2-3:1. In Lossy compression, the data is degraded, the more so the greater compression ratio achieved. Relying on people’s ability to compensate for losses, Lossy compression techniques seek a compromise between quality and quantity, exploiting the way humans perceive an image. Here, it is difficult to decide what information to discard. Lossy and Lossless techniques can be applied both spatially and temporally. Spatial compression (or Intraframe) is applied to a single frame, while temporal compression (or Interframe), looks for differences between successive frames and stores only those differences.
The many algorithms and the different ways to apply them differentiate the various CODECs. Strengths and weaknesses of the CODECs are displayed during decompression. The demands of the system are very high at this stage. Data must be received, decompressed as fast and to a high quality as is possible and transferred to the display card. Some of the most popular CODECs available are explained next.

2.3.2.1. Video for Windows

*Video for Windows* and *Quicktime* (explained later) act as containers for new video CODECs and are not CODECs in themselves. *Video for Windows* is the Microsoft’s AVI (*Audio Video Interleaved*) file standard. The AVI format only defines how the video and audio will be stored on the hard disc. It uses a technique known as *Interleaving*, where one frame’s audio is followed by that frame’s video. This technique, avoids programs from having to jump from place to place on the hard disc in order to find the next bit in the sequence. AVI does not define how the video is captured, compressed or played back. Whenever new technology is introduced (e.g. new CODECs), it can be incorporated into *Video for Windows*.

2.3.2.2. Quicktime

*Quicktime* is the Apple’s equivalent to *Video for Windows*. It is also available for Windows 95. Many CODECs support this format as well (e.g. *Indeo 3, Cinepak, Microsoft Video 1*), offering compression ratios from 50 to 60:1. Nevertheless, *Quicktime* movies play back slower on PCs than AVI files.

2.3.2.3. Intel’s Indeo

In *Indeo* the compression can take place in real time and decompression is done in software only. It is a 24-bits CODEC that dithers to 256 colours when required. *Indeo* can achieve 10:1 compression ratios. There are two versions: *Indeo 3.2* and *Indeo Interactive*. The latter is a *Wavelet* based software CODEC that enables real time interaction and control of video and graphics imagery in multimedia applications. *Wavelet* is a type of transform used during video compression and not a CODEC in
itself. *Indeo Interactive 4.2* is the latest release of the CODEC. It allows video to be played on systems that support *Microsoft Video for Windows* without adding special hardware and has been optimised for the Intel's Pentium family of processors. Moreover, *Indeo Interactive 4.2*, brings new features. For example, improved driver integration, transparency and also, improved visual quality. Local decode saves time when much of the source image does not need to be displayed. *Indeo Interactive 4.2* has been specifically designed for powerful multimedia application development, (e.g. *Indeo Interactive* incorporates unique features that make it possible to include video in interactive multimedia applications and games. For example, real time video effects). A deeper explanation of this CODEC and its new characteristics can be found in Intel (1997).

2.3.2.4. *Cinepak*

A very good image quality is offered by *Cinepak*. It is a vector quantisation based CODEC. Here, information about differences between frames of video is stored by quantifying the magnitude and direction of a pixel's movement. Decompression is very efficient and is considered to be better than *Indeo 3.2* for high action sequences, (Williams et. al, 1996a). Both, compression and decompression are performed in software and compression ratios of 10-20:1 can be obtained. *Cinepak* and *Indeo* can achieve 25 fps at 320x240 pixels and by using graphics acceleration, they can be enlarged to 640x480.

2.3.2.5. *Microsoft Video 1*

It is a single CODEC based on run length encoding and optimised for animation or cartoons. It is not as good as *Indeo* or *Cinepak*.

2.3.2.6. *Joint Photographic Experts Group (JPEG)*

JPEG is a CODEC for still image compression. It removes the redundancies in individual frames. Both, JPEG and MPEG (see section 2.3.2.7) employ a discrete cosine transform in their algorithm. *Motion JPEG* (MJPEG) is based on the same algorithms as
JPEG to create compressed frames and compression/decompression takes 1/30 of second for each frame. A problem is that various non-compatible versions exist.

2.3.2.7. Motion Picture Experts Group (MPEG)

There are two standards in MPEG: MPEG I and MPEG II. MPEG II is designed for images of high definition TV size (1.2 Mbit/second at 704x480 and 30 fps). MPEG I is designed to allow retrieval from a single speed CD-ROM (320x240 at 30 fps). Compression ratios of 30:1 to 200:1 can be obtained. MPEG is more advanced than MJPEG. Here, information in the current frame is used to predict the information in following frames (predictive calculation). Only the differences from its predictions is encoded, (Interframe). There are algorithms available to play MPEG I on Video for Windows and Quicktime. However, MPEG is a technology that still suffers from incompatibility and scalar problems. For example, "Several computer magazines are reporting on incompatibility problems between computer, CD-ROM drive and MPEG playback cards and software," (Williams et. al, 1996a, p.15). Moreover, hardware is required frequently (this is no longer true in MPEG I, which is the earlier version). Software CODECS are not yet to be entirely replaced by MPEG, thanks to new CODECS’ versions and low cost general purpose video acceleration cards. Not only the quality of CODECS like Indeo and Cinepak close to that of MPEG, but also they are considerably cheaper for encoding and playback, (Williams et. al, 1996a).

2.3.2.8. Fractal Technology

Fractal Technology is based on fractal transforms. The compression is slow, but the de-compression is very fast. One of the advantages of this technology is its scalability. For example, the video’s resolution is independent of the size of the window in which it plays, because fractal images are encoded with equations. Not only compression ratios of 100:1 have been achieved in still images but also full screen videos playing at 30 frames per second. However, Fractal Technology is not yet a mainstream technology.
2.4. A Virtual Environments Arthroscopy Training System (VEKATS)

VEKATS is a flexible system which can be used in the early and intermediate stages of the knee arthroscopy training process. It uses a computer generated model that can be easily altered in order to show unusual pathologies. VEKATS permits the development of skills such as orientation, triangulation and dexterity, simultaneously with their objective evaluation.

A simulated non-distorted Arthroscopic View is used. This view is very similar to the view seen in the operating theatre. Additionally, an Anatomical Overview can be superimposed that shows the arthroscope's position in relation to the knee, as if viewed from outside the knee. Also, live video clips may be overlaid, but actually this option only consists of a single video showing a classic traversal of a real knee. The trainee can attempt certain paths and gain orientation skills. VEKATS can help to develop dexterity in the movement through tight cavities. Training in basic triangulation is done by using an arthroscope and a probe (see section 2.1). VEKATS uses a six degree-of-freedom magnetic input device to control the arthroscope and the probe. A control panel allows the user to display the Anatomical Overview, to play a video clip, to switch between arthroscope and tool and to leave the simulation. For further description of the VEKATS' interface, refer to section 4.3.

Three main tasks are performed by VEKATS: Input / Output; Display Management and Object modelling. Interaction (Input / Output) is done through the mouse, keyboard and a magnetic input device. The Display Subsystem (Display Management) controls the playback of video clips and renders the geometrical database into the Arthroscopic View and into the Anatomical Overview. The data is stored in a shared memory area which is continually updated by the Modelling Subsystem. The Modelling Subsystem (Object modelling) operates the deformation of soft structures, bulk movement of objects and collision detection.

Many techniques have been implemented in order to make the Modelling Subsystem work and in order to improve the performance of the Display Subsystem. (see section 2.2). The graphic database consists of a set of meshed objects, in this case a list of
triangles describe the surface of each object. In order to move the objects which make up the knee (e.g. distraction, inflation with saline), a hierarchy that stores the relationships between these object was created. Voxel Based Collision Detection is used for the collision with rigid objects. Delaunay Triangulation is used for automatic mesh generation of deformable objects. Among the techniques that have been implemented to collide with these deformable objects are: the Spring-and-Dashpot method, the Reference-Frame method and the Finite Element Modelling method. The latter is the technique which is currently used. The system was developed using an object oriented approach, (see section 4.1). VEKATS is still in a prototyping stage and lacks some implementation features, for example, force feedback.

VEKATS runs in a Silicon Graphics platform and relies on data generated by three utility programs: compiler, to simplify the meshes; voxel, to create the voxel map of the rigid objects and defmain, which creates deformable object data files from the geometry data files.

2.5. Conclusion

The human knee is a complex joint composed of both soft tissue and rigid bodies. When complaints appear, the best way to make a diagnostic or a surgical procedure is by performing a form of Minimal Access Surgery (MAS) called arthroscopy. The idea of MAS is to minimise the disruption to the surrounding tissue. This kind of surgery requires the use of special instruments like the arthroscope and the probe. However, there is not a standard method for the trainee to gain the skills required to perform such a surgery (e.g. orientation, dexterity and triangulation). A part of the solution to this problem is the implementation of a Virtual Environments Knee Arthroscopy Training System (VEKATS).

The aim of the VEKATS project is to provide a cheap environment where all the skills required can be learned and assessed. Nevertheless, due to its complexity VEKATS has to be developed in many stages, each stage constituting a different but related project. In order to implement a virtual environments simulation like VEKATS (which requires realism), many techniques have to be applied. For example meshes of rigid bodies can be
created with the *Marching Cubes* technique. *Delaunay Triangulation* can be used for the automatic mesh generation required for the interaction with the soft tissue. A technique based on layering can be used to simulate the movements of the knee (e.g. distraction of the knee).

The *Voxmap Based Collision Detection* is the most efficient and accurate technique to detect collisions with rigid bodies. Nevertheless, care should be taken in the generation of both the *Voxmap* and the *Pointshell*. The *Finite Element Modelling* technique seems to be the most promising one for collision detection with deformable objects due to its accuracy, despite of the fact that the calculations performed when using this method are more computer exhaustive than those required by the *Spring-and-Dashpot* and *Reference Frame* techniques.

The importance of using the techniques named above is that a computer simulation (VEKATS) requires accurate and realistic behaviour and not just behaviour that looks nice. Additionally, video can aid understanding to the simulation, but a good video playback system must be designed and a suitable video format chosen. It is important to note, nevertheless, that the VEKATS project (as a whole) is still much a prototype that can be enhanced to create the required system.
Chapter 3: Objectives, Scope and Plan of the Project

This dissertation project aims to produce a computer simulation of an arthroscopic diagnostic procedure that will be useful for training and can be further enhanced in the future. Its development requires that an exhaustive set of tasks are performed to achieve all of the objectives. The objectives, the tasks to be performed and a definition of the scope of the project are discussed in the next sections.

3.1. Objectives

Due to the popularity of the Microsoft Windows 95 environment among users and specifically among surgeons, the VEKATS source code has to be implemented in such environment. This is necessary so that the effectiveness of the platform can be tested in the future. In order to achieve this, a good understanding of the simulations techniques used by VEKATS must be gained (e.g. representation of objects, collision detection techniques, animation, etc). Additionally, programming libraries like OpenGL, GLUT and the Microsoft Foundation Classes and programming languages like C++ and Visual C++ can be used. These are the tools available in the Computer Science Department at the University of Hull and constitute its current standard. In the future these tools may be used by other students in order to improve PC VEKATS.

The current VEKATS video system consist of a single file where a classic traversal arthroscopy is performed. The idea is to investigate, compare and evaluate the different video file formats available for Windows 95 and recommend the most appropriate to use. It is also a necessity to give the user the opportunity to interact with the system by stopping, pausing, closing, going forwards or backwards and changing the video clip. This will allow the selection of a wanted video from a set that may be used to show different pathologies, which in future projects may be added to the simulation itself.

A new design of the user interface has to be prototyped. The current VEKATS' interface may not be suitable for a training system. In order to decide this, different human-computer interface design and human factors issues in computing must be
investigated. It is also important to capture opinions from the people involved in the project (e.g. supervisor and, if possible, users) about the interface currently used and about the new possibilities in the design. This can be done by means of interviews.

If time permits, the magnetic interaction device which is being currently used should be interfaced with the PC VEKATS system. Recently, VEKATS was improved by changing the interaction device from the Immersion Probe's passive arm to the magnetic device, (Wills, 1997, personal communication). This has added accuracy and freedom to the movements and the possibility of using two instruments at the same time. Ideally, such improvements should be maintained in the implementation of PC VEKATS. However, this requires a good understanding of the magnetic device as well as the way it communicates with the PC.

3.2. Scope

In VEKATS the utility programs are used to optimise the mesh representation of the objects as well as to create different representation for the objects (e.g. Voxmap, Finite Element Modelling representation, Reference Frame representation, etc.). As these programs are not a primary requirement for the PC VEKATS system to work, they will be left as they are, since the required objects may be obtained from the ones generated for VEKATS.

PC VEKATS will focus on the same collision detection methods used in VEKATS (e.g. Voxel Based, Finite Element Modelling, etc). Other techniques used to represent objects or to animate the simulation will remain the same. However, an understanding of such techniques is required as bugs or incompatibilities can be found in the implementation process. Haptic feedback will not be included in PC VEKATS, as it is the objective of another project. The interaction devices used by PC VEKATS will be the keyboard, the mouse and, if time permits, the magnetic interaction device.

Contrasting with VEKATS, PC VEKATS is not intended to be software running on a parallel machine. The system that will be developed will run using a single Pentium processor running the Microsoft Windows 95 operating system. As the original version
of VEKATS consists of several processes communicating one to each other, in the implementation of the program on PC the software has to be changed to run in a sequential manner.

PC VEKATS will allow the same kind of movement of the knee joint as VEKATS. The current system allows only limited movement, where the separation of the interarticular surfaces can be varied to simulate distraction of the knee. Nevertheless, the ability to manipulate the knee into different orientations to aid access to some compartments of the knee cavity will remain as an enhancement that can be done.

In a real arthroscope, the lens model used adds spherical aberrations and perspective distortions into the view. However, PC VEKATS' lens model will use the same undistorted view as that used in VEKATS.

A rigorous assessment of PC VEKATS in a training environment will not be done. However, a new design of the interface will be prototyped and discussed in order to make it more appropriate. A log of the tasks performed by the user that would allow the construction of a profile of the aptitude of the trainee for certain tasks will also be left for future research.

PC VEKATS, as its name suggests, will be limited to operate upon the knee joint. Nevertheless, the system may be easily extended in the future to support other joints. As VEKATS, PC VEKATS will be limited for the moment to arthroscopic diagnostic procedures.
3.3. Plan

Figure 3.1 shows the plan for the project. Literature reviews of the different topics take place together with the other tasks. This will help to find information which is adequate for the report and for the project. The rest of the tasks of the project are outlined below and can be seen in figure 3.1.

<table>
<thead>
<tr>
<th>Task Name</th>
<th>May</th>
<th>June</th>
<th>July</th>
<th>August</th>
<th>Sept</th>
</tr>
</thead>
<tbody>
<tr>
<td>Project topic analysis (ended April 21st)</td>
<td>5</td>
<td>12</td>
<td>19</td>
<td>25</td>
<td>2</td>
</tr>
<tr>
<td>Literature Revision</td>
<td>9</td>
<td>16</td>
<td>23</td>
<td>30</td>
<td>7</td>
</tr>
<tr>
<td>Review VEKATS</td>
<td>14</td>
<td>21</td>
<td>28</td>
<td>4</td>
<td>11</td>
</tr>
<tr>
<td>Implementation and understanding</td>
<td>18</td>
<td>25</td>
<td>1</td>
<td>8</td>
<td>15</td>
</tr>
<tr>
<td>User interface</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Implementation of video playback system</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Implementation of the magnetic input device</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Performance tests</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Writing</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Presentation to industrial panel</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Final revisions and submit of the project</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 3.1. Project Plan

A review of the VEKATS system is necessary prior to the implementation on PC. This involves a number of interviews with the people involved in VEKATS (e.g. this projects' supervisor, UNIX VEKATS' author). Demonstrations of the current system are also required at this stage.

The implementation of the code on PC involves tests with simplified version of the system and familiarisation with tools like OpenGL and Visual C++. Here a deep understanding of the VEKATS' source code has to be achieved.

The design of the interface is one of the most challenging tasks. It first involves the research of different issues in human-computer interaction (e.g. Consistency, etc). Next, different experiments on interaction techniques have to be done (e.g. dialogue boxes, mouse interaction, etc). Finally, the hardest stage of this task is the implementation of
the prototype. This involves interviews with the supervisor and, if possible, with the users.

The implementation of the video playback system is not expected to take too long. First, the different techniques are investigated and tested. Finally, the implementation is done. If time permits, the magnetic interaction device would be connected just before the presentation of PC VEKATS to the industrial panel.

Performance tests are done when the system is practically finished. As it can also be seen in figure 3.1, a reasonable time is assigned to the writing of the document. Finally, the dissertation should be finished by the week of the presentations with the industrial panel. Only, minor revisions and corrections should be required before the work is submitted.

3.4. Conclusion

The implementation of a PC based VEKATS, with the re-design of the user interface and the video playback system is valuable because it will constitute an improved implementation of VEKATS and a starting point for VEKATS on PC. This improved version will be suitable for extension to a full open learning system in the future. Such a learning system will complement the arthroscopic surgeon training process, allowing objective evaluation of the performance of the trainees in a non-destructive simulated environment.
Chapter 4: Design

This chapter starts with a description of the class structure of PC VEKATS. Next, the way PC VEKATS works is explained. Finally, a section describes the design of the interface and a set of conclusions is given. In each of these sections, the design of VEKATS is always contrasted with the design of PC VEKATS, and the main differences are highlighted.

4.1. Class Structure of PC VEKATS

Due to the complexity of the PC VEKATS' Booch diagram this section has been split into three different subsections referred as *Classes that build up the interface*; *Classes that build up the arthroscopic simulation* and *Classes that build up the Finite Element Modelling objects*. Each of these subsections discusses a different part of the class structure of PC VEKATS.

4.1.1. Classes that Build Up the Interface

The interface of VEKATS was created using *ViewKit* for X Window while PC VEKATS uses the *Microsoft Foundation Classes* for Windows 95. Figure 4.1 shows the VEKATS' design surrounded by a circle. In contrast, the rest of the image shows the PC VEKATS' design. This figure highlights the main difference between the class structure of VEKATS and PC VEKATS.

In VEKATS an *Application* class' object is responsible of creating an application and by using a *Main Window* class' object it creates the main window. The constructor of the *Main Window* class calls the constructor of the constructor of the *Bulletin Board* class. This constructor then starts running the VEKATS simulation. The *Bulletin Board* class also creates the control panel and processes its events (see section 4.3).

PC VEKATS, in contrast, uses the class structure generated by using the *Application Wizard* of Visual C++. As shown in figure 4.1, the *Application* class creates the
application and uses the Frame Window, View and Document classes. They are referred in the Application::InitInstance function. The Frame Window class creates the main window of the application. The characteristics (e.g. size, style, title, etc.) of the windows can be defined inside the PreCreateWindow member function of this class. The Document Class serves as the connection between documents, frame windows and views. It does not have to be changed for any reason for the case of PC VEKATS.

Figure 4.1. Classes that Build Up the Interface
The top right corner of figure 4.1 shows the View class. In this class, the PreCreateWindow member function is used to define the styles required by OpenGL (e.g. WS_CLIPSIBLINGS, WS_CLIPCHILDREN). The OnCreate member function calls the Init member function which initialises the palette and OpenGL (this task was done by the Canvas' constructor in VEKATS). The OnDraw and OnTimer member functions call the DrawScene member function. The View::DrawScene function uses an object of the Simulation class. Here, one iteration of the simulation loop is run and displayed and then the performance is evaluated. The View::OnSize function calls the Arthrocanvas::redraw function, which scales the output to the new window size.

All of the interactions (e.g. mouse events, keyboard events, choices, etc.) take place in the View class or in the Control Panel class (or Instrument Panel class), which is derived from the Dialogue Box class. The interaction can happen usually by communicating with the Simulation class or by communicating with other MFC subclasses (e.g. Menu, Radiobutton, etc.). In VEKATS most of the interaction was done by the BulletinBoard class using the functions of the Arthrocanvas member of the Simulation class. In PC VEKATS most of those callings have been moved to the Control panel class (see section 5.4).

Every dialogue box used in PC VEKATS is implemented with a class. Every time a dialogue box has to be displayed the View class' object calls the respective Dialogue Box class' object. Whenever the option to show a video clip is chosen the Control Panel class uses an object from the CWnd class. The CWnd class takes a MCI Window (or multimedia window) that displays the video.

4.1.2. Classes that Build Up the Arthroscopic Simulation

The Booch diagram in figure 4.2 shows another part of the design which is described next. The Simulation class has an Object Hold Info member which is used to record the deformable objects' status. Besides this, in the Simulation class there are a number of bones that belong to the rigid Object class, (e.g. femur, tibia, cut-down version of femur, cut-down version of tibia, fibula, patella, poplit, pouch, suprapatella pouch, scope, arthroprobe and overlay). Deformable objects of type Finite Element Modelling Object
class are also defined here. They represent the soft tissues of the knee. Objects of the Vector class are used to keep the position of the probe and the arthroscope. Additionally there is an Arthrocanvas drawing canvas for the simulation.

The Simulation’s constructor assigns a dynamic memory area to keep all the objects. The destructor clears this memory. The Simulation::Initialise function, initialises the writing canvas and reads the meshes to be used in the simulation after reading the configuration file by calling the Simulation::ReadConfiguration function. The Simulation::SimLoop function starts the simulation (in VEKATS) or runs one iteration of the simulation (in PC VEKATS) and uses the Simulation::CheckCollision function. Finally, in the Simulation class, there is a number of numeric member variables and other functions that handle with the maths and the status of the simulation.

The next class from figure 4.2 to be described is the Canvas class. It defines a basic drawing canvas. Only rigid objects of the Object class can be drawn on this canvas. It includes member functions and variables that can be used to get the frame rate. The Canvas::Render function is called once every rendering cycle and indicates what to draw, what viewpoint to use, etc. The Canvas::Draw function draws an object’s polygons. The method takes ‘p’ polygons and normals compiled into a single list, ‘db’. The object is drawn with an ‘r,g,b’ colour and a transparency ‘t’. Besides this, ‘X,Y,Z’ translations and a ‘rot’ rotation are applied to the object with a scaling factor ‘s’.

Another overloaded version of Canvas::Draw does the same, but uses an unoptimised list of vertices and their polygonal connections. The Canvas::DrawL function draws an object from the optimised polygon list ‘db’ without shading it. One difference with VEKATS is that in PC VEKATS the Canvas’ constructor neither creates a window nor initialises OpenGL.

The Deformable Canvas class is derived from the Canvas class and adds functionality that allows to draw deformable objects. The new overloaded Draw function draws the object’s deformation structure as given by the ‘n’ and ‘c’ lists that belong to the classes Deformable Node and Deformable Connection respectively. The Arthrocanvas class, which is derived from the Deformable Canvas class, adds functionality to the canvas which is specific to an arthroscopic canvas. For example the MakeOverlay member
function is defined here. Other functions that are called from the Control panel class when an event occurs are also defined here (e.g. Arthrocanvas::DoSep, which is called when the knee is injected with saline). For more details on this refer to section 5.1.

Figure 4.2. Classes that Build Up the Arthroscopic Simulation

The bottom left of figure 4.2 shows the Object class' hierarchy. The Object class is the basic rigid polygonal object. Three vectors define the centre, rotation, and translation of the object. One float defines the scaling. The Object::Read function reads in a .GLC file to create an object. The Object::VoxRead member function reads in a .VOX file to
create an object’s voxel image. The `Object::TestObject` function takes a line ‘f - b’ and checks it against a voxel map to detect a collision with the object and return the collision point. The `Object::Draw` function call the `Canvas::Draw` function with the compiled polygon information, ‘db’. The `Object::DrawLists` function calls the `Canvas::DrawL` function.

The `Deformable Object` class is derived from the `Object` class. It implements the generic deformable object functions and contains methods which are identical across all types of deformable objects (e.g. `Draw`). Several of the methods of the `Deformable Object` class are empty. It is more an abstract class. This class contains a `Deformable Node` class’ list of nodes of the `Deformable Object`, a `Deformable Connections` class’ list of connections between nodes and the type of `Deformable Object` (e.g. `Reference Frame`, etc).

The function `Deformable Object::DrawLists` calls the `Deformable Canvas`’s `Draw` functions to draw the object. The `Deformable Object::TestsObject` function tests if the line ‘f - b’ intersects the object. `Deformable Object::Update` updates the deformation of the object (if a flag has been set to TRUE). `Deformable Object::Reset` restores the deformation back to the initial position. `Deformable Object::GetForce` returns the force in the deformation structure at node ‘n’. `Deformable Object::Release` releases all constrained nodes and resets all dynamic constraints. `Deformable Object::Hold` returns the nearest node to a position ‘p’ and is used in conjunction with `Deformable Object::Constraint` to find the point of interaction with the `Deformable Object`. `Deformable Object::Constraint` constraints node ‘n’ to either position ‘p’ or force ‘f’ depending on the deformation model being used. `Deformable Object::CreateMesh` uses tessellation methods (e.g. Delaunay Triangulation) to create a mesh that is going to be used as the object’s deformation structure. Additionally, there are other member functions and variable which are used to build the various data structures related to deformable object modelling.

The `Spring Object` and `Reference frame Object` classes (see figure 4.2) are derived from the `Deformable Object` class and implement the `Spring-and-Dashpot` deformation method and the `Spring-and-Dashpot` deformation method with skeleton `Reference Frame` respectively. Finally, the `Finite Element Modelling Method Object` class extends
the *Deformable Object* class and is explained in the next section. The *Vector* class shown as well in figure 4.2 is a general 3D vector class. It supports the standard mathematical operations on vectors, (e.g. sum, dot product, magnitude, etc.).

### 4.1.3. Classes that Build Up the Finite Element Modelling Objects

Figure 4.3 shows the Booch diagram of the class structure used by the *Finite Element Modelling Object* class.

![Diagram](image)

*Figure 4.3. Classes that Build Up the Finite Element Modelling Objects*
The Finite Element Modelling Object class has a Matrix 2D member which contains the stiffness matrix, a Matrix 1D which contains the force and displacement vectors and a Constraint List class' list. The member functions are those from the Deformable Object class and are usually overwritten. The Matrix 1D class defines a dynamically sizeable one dimensional matrix with the usual operations (e.g. multiplication, assignment, etc). The Matrix 2D class defines a dynamically sizeable two dimensional matrix with the usual operations (e.g. multiplication, identity, determinant, etc).

4.2. How PC VEKATS Works

VEKATS was designed to run in a parallel machine. During its execution three processes are running. The Rendering Engine, the Object Modelling Engine and the Control Panel Monitor. The Rendering Engine, which requires data from the other two processes, consist of the Renderer Module. It also takes data from the Configuration Module at the beginning. The Control Panel Monitor handles all user interaction performed in the control panel window. Finally, the Object Modelling Engine includes the Object Modeller, Collision Detection and I/O Magnetic Device modules. Other modules like the Voxeliser, Rigid Object Compiler and Deformable Object Compiler are run before to prepare the data for the simulation. Figure 4.4 shows a diagram of this system.
PC VEKATS, in contrast, is designed to run using a single Pentium processor running the Microsoft Windows 95 operating system. The system takes advantage of the multitasking features of the operating system. For example, the event handling (e.g. interaction events, system events, etc.) can be treated 'concurrently' while the simulation works. However, processes like the Rendering Engine and the Object Modeller cannot run in parallel. Due to this fact, the design had to be sequentilised. In PC VEKATS one iteration of the Object Modeller runs each time after the Rendering Engine is run.

4.3. Design of the Interface

"The visual design of a user-computer interface affects both the user's initial impression of the interface and the system's longer-term usefulness. Visual design compromises all the graphic elements of an interface, including overall screen layout, menu and form design, use of colour, information codings, and placement of individual units of information with respect to one another. Good visual design strives for clarity, consistency, and attractive appearance," (Foley et. al., 1996, p.148).
Figure 4.5 shows the current VEKATS' interface. After initialisation, the system shows the main window and the Control Panel window. The default position and size of these windows can be changed if required. The Control Panel allows the user to modify aspects of the simulation. The Quit option quits the simulation. The Release option releases a soft object if the tool has grasped it. The Video option plays a short video clip of an arthroscopic traversal in the bottom right hand corner of the screen. The Separation sliding bar modifies the distraction of the knee.

The Arthroscopic View window (see figure 4.5) consists of a circular view-port through which the anatomy of the knee is visible. It can be re-sized and re-positioned. If the magnetic input device is controlling the arthroscope, the view will move and rotate when the device is used. An optional Anatomic Overview (see figure 4.5) can appear at the top right hand corner. It is enabled or disabled through a configuration file. This view can be zoomed in and out by moving the mouse with the left button pressed. It can also be rotated by moving the mouse with the right button pressed. The Anatomic Overview intends to aid orientation within the knee.
Other aspects of the VEKATS' system behaviour are controlled by the configuration file. It not only controls the availability of an Anatomical Overview. It also controls the behaviour of the probe (e.g. warn of a collision with a soft object, grasp the soft object), the field of view of the arthroscope (30° or 70°), the filename of the tutorial video and the file names with the mesh representation of the objects.

The quotation in the first paragraph of this section highlights the importance of a good interface. Many authors like Sutcliffe (1988), Foley et. al. (1995) and Jones and Okey (1997) have attacked the aspect of interface design. Most of them agree in the necessity of a good interface and produce a set of guidelines that can help to accomplish it. Despite of the fact these guidelines have been named differently by the different authors, the reality is that they can be grouped into four big areas as follows. Consistency, or the ability of a system to look, feel and act the same way throughout. Feedback, or the ability of a system to show a response to the user's interaction. This includes things like highlighting the current selection, immediate changes or responses in the system, indication of values, etc. Error Manipulation, or the ability of a system to minimise error possibilities or include error recovery. Grouping, or the ability of a system to minimise the user's effort to memorise how to operate it.

The design of the PC VEKATS' interface is explained next, together with a detailed description of how different guidelines have been applied to this design.
In order to maintain the consistency in the system, PC VEKATS and all of its interface elements (e.g. menu, dialogue boxes, windows, etc.) have to behave like a Windows 95 application. Figure 4.6 shows the current PC VEKATS' interface. It shows the Instrument Panel at the top, the Control Panel at the right, the Video Window at the bottom and the main window with both the Arthroscopic View and Anatomic Overview at the top right side. When these windows are first called they appear with a default position and size. However, all of them can be moved, minimised or closed. Only the Main Window and the Video Window can be freely re-sized, since the Control Panel and the Instrument Panel are dialogue boxes rather than proper windows. The menu is placed at the expected location for a standard Windows 95 pop-up menu.
The menu applies the *Grouping* concept in a logical manner. All of the option can also be accessed with key shortcuts. The *Simulation* option allows to open a simulation or pathology or to exit PC VEKATS. The View option allows to show or hide the control and instrument panels. The *Help* option should in the future include some kind of on-line help for the system. Every time an option is chosen, feedback to the user is given in different forms. For example, an option to be chosen is always highlighted in blue. Moreover, the choices in the View option are checked (✓) or not depending on when the respective panels are currently being displayed. Finally, a choice in the menu is evident since a window or dialogue box is displayed immediately after it.

![Open Simulation](image)

*Figure 4.7. Open Simulation*

The *Simulation: Open* option displays a modal dialogue box (see figure 4.7) that requires immediate response (as expected in this kind of situation). The user can either cancel the choice or choose a simulation via a combo box. This dialogue box appears in the current PC VEKATS' interface prototype and can be used in the future to include more than one pathology.
Figure 4.8 shows the PC VEKATS' Control Panel. The Control Panel follows the standard Windows 95 layout and all of its control behave in manner consistent with any other Windows 95 application. Default values are given, so the user knows the current situation in the program. The concept of Grouping is applied, as described in the following paragraph.

The Video Information group deals with the video to be displayed. Here the user can choose from a list of clips available for the current pathology by using a combo box or by typing in the name of the video. The system will indicate if the video name does not exist. It is also possible to play more than one video clip.

In the Distraction of the Knee group, a value can be entered by clicking up or down on the arrow icons or by typing in the edit box. Feedback is obtained since the knee is separated while the user changes the value. Error Prevention is given by indicating the limit values for the distraction of the knee. Error Recovery is provided in two ways.
First, when an invalid value is entered, it is automatically changed to the maximum (if the value entered was too high) or to the minimum (if the value entered was too low). Besides this, if the user enters a valid but not desired value he/she can regain the original position with the reset button.

In the Anatomic Overview group, the check buttons are used to turn it on or off, providing extra feedback to the user about whether the overview is being displayed or not. These buttons also indicate that only one of the options can be chosen at a given time, providing some kind of Error Prevention. The arrow icons simulate a virtual joystick. This idea was suggested by the project’s supervisor. The most common options (e.g. rotation on Y and Z) were grouped together, while the less common (rotation on X) was separated. The zooming of the overview uses the same kind of arrows. Indication of the rotation axis and the zooming direction is given by using extra text next to the icons. This group also has a reset option.

The Arthroscopic Simulation group is where the configuration commands are. An advantage of having this group is that the configuration of PC VEKATS can be changed interactively as the simulation goes and does not rely on the configuration file only (like in VEKATS). The same kind of Feedback, Error Manipulation, Consistency and Grouping issues have been applied here. Additionally, the check boxes are used to indicate that many of the objects can be chosen at the same time in the View subgroup. The ability of selecting different parts of the anatomy of the knee can improve understanding.

In PC VEKATS, the Feedback for a collision with soft tissue can be chosen to be a warning, a grasp or a cut on the image. Collisions with rigid objects always cause the background of the main window to become red.
The Instrument Panel (see figure 4.9) allows the user to control, move and rotate the arthroscope or the tool. As it is seen, the Grouping of common elements is applied again. The interaction elements are similar to those used in the Control Panel keeping Consistency within the program. Immediate Feedback is obtained as the instruments in the Arthroscopic View are moved together with the virtual joystick in the panel.

Figure 4.10 shows the Video Window. Contrasting with VEKATS, PC VEKATS allows much more control over the video clip the user is playing. For example, he/she can
change the speed, size and even the video clip. It is easy to pause, go forward or backwards. This is thanks to the use of a multimedia window which is a standard in Windows 95, together with improvements for multimedia interaction. Feedback is given as the options are chosen (e.g. the selection are highlighted, the values chosen are shown). The icons used can easily become natural for the user and are standards used, for example, in the Windows 95's Media player application. Additional information about the number of the frame that is currently being played and about the name of the video is displayed in the title bar of the window.

The main window is never obscured by any of the others. The guidelines proposed were followed as close as it was possible and considerations for the different kinds of interaction proposed by Foley (1995) were also taken into account. For example the Selection task follows the Windows 95 standards (e.g. Grouping of Elements, Menu Placement, Dialogue Boxes, etc). The Quantisation task is made easier by providing default values and familiar interaction objects (e.g. arrow icons, slide bars, etc). The Positioning task is done mainly with the mouse (e.g. menu selections, window movement, etc). New positions in the simulation can also be reached by using the virtual joysticks or, if time permits, the magnetic interaction device. Finally, the Text interaction task is natural and 'Windows 95 like' in all the edit boxes present in the interface.

4.4. Conclusion

The object oriented structure of the VEKATS' design facilitated the extension and change to Windows 95. For example, the Bulletin board classes were changed to the Microsoft foundation Classes without major problems. Moreover, Some member variables or functions could be easily changed, added or removed. However, the design of PC VEKATS was kept as close as possible to the original VEKATS design.

The construction of Booch diagrams allowed the author of this project to gain a deeper understanding of VEKATS. Through the creation of them it was possible to understand the impact of some changes done to the PC VEKATS' design (e.g. the communication of the Control panel class with the Simulation class). The object oriented structure of
VEAKTS was definitely a characteristic that allowed the author of PC VEKATS to progress rapidly.

Many guidelines have been found and applied in interface design. Frequently these were discussed by means of interviews with the project's supervisor. The PC VEKATS interface is an improvement over that found in VEKATS because it does not simply rely on the initial configuration file. Moreover, the characteristics of the simulation can be changed interactively. The new interface offers Feedback, Consistency, Grouping and Error Manipulation, making it easier to use and to learn. Contrasting with VEKATS, the actions that can be performed on the system are now self-evident. For example, in VEKATS there is no hint for the user about the possibilities of rotating and zooming the Anatomic Overview. Finally, the PC VEKATS' interface can be easily extended and improved.
Chapter 5: Implementation

This chapter explains how PC VEKATS was implemented. First a description of the implementation of the system itself is given. Next, the implementation of the video playback system is discussed. Later, a section explains the implementation of the magnetic interaction device. Finally, the implementation of the interface is described and a set of conclusions is given. Appendix 1 explains in which files are implemented the different classes and gives information about where to find the programs that were used in order to implement PC VEKATS.

5.1. The Implementation of the Code on Windows 95

The implementation of PC VEKATS was based on code from an example program by Nguyen (1995) and on code provided by Logan (1997). The latter is the original VEKATS code, the former is an application generated with the Visual C++’s Application Wizard, but it was modified to permit OpenGL functionality without losing the advantages of the Microsoft Foundation Classes (MFC).

The implementation of PC VEKATS was divided into different stages. First, the test of simplified versions of VEKATS. Next, the implementation of the whole system on PC. Later, the coding of the video playback system (see section 5.2). At this stage, the implementation of the designed interface was required (see section 5.4). Finally, the connection of the magnetic interaction device (see section 5.3).

The simplified versions include: a program to display and spin a rigid Object in a Canvas, a program to display and spin a Deformable Object (e.g. Reference Frame, Finite Element Modelling) in a Deformable Canvas and a program to display an object allowing some interactivity. The objective of this stage was to become familiar with VEKATS and with the tools used to implement PC VEKATS.

The source code for these simplified versions was written in C++ and uses the graphic utility libraries (GLUT) for OpenGL. This code was not platform dependent and could
be successfully implemented on LINUX and on Windows 95. The task was reasonably
straight forward and the kind of problems that arose were related with familiarisation
with the different compilers and environments. For example, if a program is to be
compiled in Visual C++ it must have the .CPP extension and not the .C one used in
VEKATS. Besides this, all the OpenGL libraries must be input to the linker and placed
in the right location in both LINUX and Windows 95.

The implementation of the program that allows interactivity is explained in more detail
in section 5.4. This program is able to use the MFC and OpenGL at the same time.

The implementation of the whole VEKATS system on the PC was divided into three
sub-stages. First, it was necessary to comment out all the code that had to do with
interaction (including that of the magnetic interaction device) and with timing (e.g.
Canvas::GetFrameRate), since this code was platform dependent. Second, the objects
required for the simulation had to be displayed. Third, the timing functions were
implemented. The implementation of the rendering functions and the functions related
with the simulation, as well as the major changes made to the original code and the
problems found, are explained in the following sections.

5.1.1. The View Class

Some of the functionality of the View class was changed, (the input and output take place
in this class). For example, the View::DrawScene function calls first the
Arthrocanvas::Render function of the Canvas object that belongs to the Simulation
‘Sim’. Having done that, one iteration of the simulation loop is called. Since the system
now runs sequentially, the Simulation::SimLoop function is not an infinite loop anymore
(like in VEKATS) but a single iteration. The View::DrawScene function is called every
time step through the View::OnTimer function or every time the main window is
changed through the View::OnDraw function. In PC VEKATS, the initialisation of
OpenGL was moved from the Canvas’ constructor to the View::Init function. However,
to make the initialisation process work, the calling to the glutInitDisplayMode function
was left in the Canvas’ constructor. Problems can appear if the ‘n’ parameter (added by
the author) in the View::PreCreateWindow function is taken out.
5.1.2. The Simulation Class

This class starts the simulation of the arthroscopic procedure. In VEKATS, the constructor function of the Simulation class creates a shared memory area, represented by a file. This area is used by the Simulation's member objects to exchange information. However, such an area was not required in the sequential PC VEKATS. Here, the constructor assigns a space of dynamic memory to the Simulation's objects. In contrast, the destructor clears this memory. In PC VEKATS, the constructor also calls the Simulation::Initialise function which is responsible for reading the configuration file.

5.1.3. The Arthrocanvas Class

This class implements a canvas were all the objects involved in the arthroscopic procedure are drawn. In PC VEKATS, the Arthrocanvas::Render function presented some problems, which are now described:

The anatomy of the knee was not displayed. This problem was caused because the magnetic interaction device was not connected. In this case, the arthroscope was pointing to the wrong direction. A solution was to change the positions indicated in the gluLookAt function.

The overlay was not displayed. This problem was caused because the glClearColor function was called after the Arthrocanvas::MakeOverlay function, since this caused all the buffers to clear (e.g. stencil, colour, etc). A solution was to invert the order of the calling to the functions.

Trying to display the Deformable Objects caused the system to crash. This problem was caused by one bug found in the VEKATS' source code. In this bug, an upper limit in a loop that was used in the Deformable Object::Draw function went outside the range of the array of vertices. The solution implemented on PC VEKATS changed the value of the limit that caused the problem.
A test program that implements the timing functions was created. The technique used here to get the elapsed time since the beginning of the simulation was used in the PC VEKAT's implementation of the Canvas: :GetFrameRate function. The technique is explained now. The TIME.H file has to be included. Two variables of type clock_t can be defined. One keeps the starting time and the other one the current time. A value can be assigned to them by calling the clock function in the appropriate moment (e.g. for the starting time at the beginning of the simulation and for the final time in every time step). The elapsed time in seconds can be calculated by applying equation 5.1 (shown below). Having done this, it is possible to get the frame rate.

\[
E_t = \frac{C_t - S_t}{CPS} \quad (5.1)
\]

Where:

- \( E_t \) = Elapsed time.
- \( C_t \) = Current time.
- \( S_t \) = Starting time.
- \( CPS = \text{CLOCKS\_PER\_SEC} \), (which is a constant value used by Windows 95).

### 5.2. The Video Playback System

Some of the criteria used to decide the recommended video format were Compression Level, Quality of Compressed Video and Compression / Decompression Speed. These three aspects are analysed in section 6.1. Different formats were not chosen for different reasons. For example, Quicktime format performs worse on PC than Video for Windows (see section 2.3.2.2). MJPEG is just a primitive version of MPEG (see section 2.3.2.7). The MPEG format suffers from incompatibility problems and is dependent on hardware (specially for latter versions), which make it more difficult to update. Finally, Fractal Technology is not yet a leading technology, (see section 2.3.2.8).

For the implementation of PC VEKATS, Video for Windows (AVI format) was chosen because it is easily upgraded to new software CODECS (e.g. Intels' Interactive Indeo...
4.2), (see section 2.3.2.1). It is also the more suitable format for programming in Visual C++, since this compiler comes with a set of functions and libraries that are able to handle AVI.

When a format like AVI is used, it is a necessity to decide which CODEC to install, (see section 2.3.2.1). CODECS different from Cinepak or Indeo were discarded since they clearly perform worse (see section 6.1). Cinepak performs better in fast action scenes and according to the results seems to be a bit better than Indeo 3.2, (see sections 2.3.2.4 and 6.1). However, an arthroscopic surgery is not a fast action scene and Interactive Indeo seems to be a more promising CODEC, (see sections 2.3.2.3 and 6.1). This was the CODEC suggested for PC VEKATS.

For the implementation of the video playback system the following recommendations given by Williams et. al. (1996a) were taken into account: the top left corner of the video window must be placed on horizontal pixel co-ordinates that are divisible by 4, since Microsoft states that performance can be up to 50% worse for unaligned video. The window is originally shown at the resolution of the original file frame size, since scaling the video window is heavy on processing power.

As with the rest of the code. The implementation of the PC VEKATS’ video playback system started with a simplified program that was integrated with the rest of the system. For the implementation of this video playback system the Multimedia Control Interface (MCI) window was suggested and adopted. Options like the Visual C++’s Animation Control and the MCI’s Set of Commands were not used. The former was not used since this control does not offer a good interface to interact with the video, and moreover, the kind of video that can be played in such a control is restricted (e.g. no sound). The latter option was not used since the MCI’s Set of Commands are 16-bit based functions that does not take advantage of a 32-bit operating system like Windows 95. The usage of the MCI Window is explained next.

The MCI Window is a new 32-bit multimedia window supported by Visual C++. It includes all the interaction objects required in a standard Windows 95 interface, allowing functions such as play, stop, change and re-size to be performed on the video stream.
However the implementation of this window was not a trivial task. All the functionality for this object is defined in the VFW.H file and the libraries WINMM.LIB and VFW32.LIB have to be input to the linker. A multimedia window is created with the MCIWndCreate function. Since this function requires the instance of the application as a parameter, it has to be obtained from the 'cs' structure used in the Frame Window::PreCreateWindow function. Besides that, in order to be shown and in order to have control over it a MCI window has to be attached to a CWnd object. After this is done, the CWnd::MoveWindow and CWnd::SetWindowPos functions can be used to locate the window in the right position and to make it appear always on the top.

5.3. The Magnetic Interaction Device

In order to connect this device, one of the PCs had to be moved to the Virtual Environments laboratory. Since this did not happen with enough anticipation, it was no possible to connect the magnetic interaction device to PC VEKATS. However, the instrument panel (see section 4.3) deals with the interaction with the arthroscope and the probe, allowing demonstrations and usability of the system even when the magnetic interaction device is not present. The author of this report recommends to have a look at the GetTrackerInfo and gluLookAt functions in the Canvas::render function in order to accomplish this objective.

5.4. The Interface

In VEKATS the interface was created using ViewKit for X Window and many of the interaction callback functions in the Bulletin Board class call member functions of the Arthrocanvas class. This is a platform dependent implementation that had to be changed. There were two choices for implementing the PC VEKATS' interface. The first option was to implement a system using GLUT. This would have created platform independent software. However, many of the visual characteristics provided by the MFC would have been lost and many of the visual elements in the current implemented interface would have been impossible. Taking this into account, together with a statement from Kilgard (1996) that says that the popup menu facility offered by GLUT cannot be used to create a full-featured user interface, the MFC were used to implement
the PC VEKATS' interface. The idea when using the MFC is the same as that of when using ViewKit, but now the callback functions are members of the Control panel and View classes.

The implementation of the interface started by using a simplified program where some interaction events were mixed with OpenGL code. In a program like this, the interaction can be added by using the Visual C++'s Class Wizard. Every time a callback function for an interaction event is required, it is added as member function of the View or the Dialogue box classes. The code inside these function usually calls the member functions of Arthrocanvas.

In both VEKATS and PC VEKATS, the Arthrocanvas::Render function is responsible for checking the value of flags. This helps to decide what is to be displayed or changed in the simulation (e.g. Anatomic Overview, fog). However, in PC VEKATS the display of elements like the Control panel or the Video Window is controlled directly from the View class. It is important to note that the code for controlling the fog in the simulation did not exist in VEKATS (it was added to add enhance the realism). Also, the code that controls the magnification factor and rotation of the Anatomic Overview was changed, since it is now done by using a virtual joystick and not by mouse movements.

The PC VEKATS' interface uses elements like dialogue boxes, menus, etc. All these elements were created with the Visual C++'s Resource Editor. The dialogue boxes can be called from the View class by calling the Dialogue Box::DoModal function for Modal windows or the Dialogue Box::Create for No Modal windows. No Modal windows allow the user to continue using other parts of the program while they stay visible on the top. In contrast, Modal windows require an answer before allowing the user to continue, (Kruglinski, 1993). Every time a Dialogue Box is created a class for this is created and callback functions should be programmed for every interaction object placed in the Dialogue Box. Whenever such a function is required, it is added as a member function of the respective Dialogue Box. It is out of the scope of this project to explain how every control (or interaction element) in the Dialogue Box can be used, for a better explanation on this refer to Kruglinski (1993) or to any other Visual C++ book. However, a difficulty that occurred in the implementation is explained next.
One of the problems that appeared when the controls of the dialogue boxes were programmed has to do with the edit boxes. In order to make a ‘Windows 95 like’ application, these edit boxes have to accept an input value followed by the Enter key. However, this behaviour was not easy to program. First, an invisible button that absorbs the Enter key (stopping the dialogue box from disappearing) and controls the changes in the edit boxes was placed and programmed in the dialogue box. Besides this, as the edit boxes have been programmed, they are able to tell the Absorb Button which edit box was the last to be changed. Only the changes in the edit box where the Enter key is pressed are taken into account.

Another interface matter is now explained. The default size, title and style of the main window of a Windows 95 application programmed in Visual C++ must be defined by altering the fields of the ‘cs’ structure of the Frame window::PreCreateWindow function. This function indicates to the program the characteristics of the applications’ window before it is created.

Despite of the fact it can be said that the windows used by PC VEKATS (e.g. Main Window, Video Window, etc) are consistent within the application and with Windows 95 in the sense that they were created to behave in a ‘Windows 95 like’ way, a small inconsistency problem still remains in the implementation, it is explained next. Due to difficulties the author had, the Anatomic Overview was not placed in a separate window. As it is seen in Figure 4.6 (see section 4.3), the panels are windows (instrument and control panel), the video is displayed in a window, etc. However, the Anatomic Overview stays together with the Arthroscopic View. This fact clearly contrasts with the rest of the PC VEKATS interface design and, for instance, with Windows 95, since the Anatomic Overview does not have the properties that would be expected from a standard window in Windows 95. It is expected and suggested that future versions or PC VEKATS include a solution to this problem.

5.5. Conclusion

The changing of a source code from one platform to another may seem to be a trivial task, but, many problems may arise in the process. For example, becoming familiar with
the different compiler and environment styles takes some considerable time. Moreover, finding bugs in other’s people code is a very time consuming task. Finally, characteristics specific to each platform have to be learned and identified (e.g. timing functions, interface extensions).

The AVI video format still provides the best playback system standard for a platform like Windows 95, since it is naturally programmable with compilers like Visual C++. AVI gives good performance, is easily updated and does not rely on hardware.

Visual C++ is a powerful tool. It allows the programmer to include many state-of-the-art interaction elements. Moreover, class manipulation becomes a direct and almost automatic process. However many details have to be taken into account (e.g. the usage of the different controls, system dependent code, etc). In contrast, the use of Visual C++ together with OpenGL is a much more complex task. In fact the author of this project had to modify a sample source code and did not manage to create two OpenGL windows at the same time. This last problem stopped the creation of a separate window for the Anatomic Overview. Nevertheless, OpenGL and GLUT are still ideal tools for 3D graphics manipulation, despite of the fact that they are not recommended for interface creation.
Chapter 6: Results

This chapter first gives a comparison of different video formats and CODECS. Next, an evaluation of the performance of PC VEKATS using two different graphic cards (e.g. STB Velocity, Matrox millennium) is given. The chapter ends with a set of conclusions.

6.1. The Video Playback System

The following results taken from Williams et. al. (1996a) can be used to compare different CODECS and video file formats on PC. The experiment was done taking a video sequence which contains several scenes with different content (scenes with little change, rapidly changing scenes, scenes with multiple content and special effects, etc). This sequence was captured raw and using real-time compression at various frames rates and window sizes, and compressed using a variety of different quality settings and CODECS. The Creative Labs Videoblastter RT 300, which performs-real time Indeo compresion and captures AVI, was used in the trials.

In the tests 30 seconds of Intel Raw video were captured at 25 frames per second with a window size of 320x240. The original file size was 70.4 MB. All files are 24 bit colour except those created with Microsoft Video I as this CODEC only supports 16 bit colour.
<table>
<thead>
<tr>
<th>Quality setting</th>
<th>CODEC</th>
<th>Size (MB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>100% quality key</td>
<td>MS Video 1</td>
<td>70.1</td>
</tr>
<tr>
<td>frame every frame</td>
<td>Indeo 3.2</td>
<td>13</td>
</tr>
<tr>
<td></td>
<td>Indeo Int.</td>
<td>12.4</td>
</tr>
<tr>
<td></td>
<td>Cinepak</td>
<td>18.7</td>
</tr>
<tr>
<td>0% quality key</td>
<td>MS Video 1</td>
<td>0.61</td>
</tr>
<tr>
<td>frame every 10 frames</td>
<td>Indeo 3.2</td>
<td>2.65</td>
</tr>
<tr>
<td></td>
<td>Indeo Int.</td>
<td>1.79</td>
</tr>
<tr>
<td></td>
<td>Cinepak</td>
<td>18</td>
</tr>
<tr>
<td>CODEC defaults for quality and key</td>
<td>MS Video 1</td>
<td>5.17</td>
</tr>
<tr>
<td>frame rate</td>
<td>Indeo 3.2</td>
<td>2.89</td>
</tr>
<tr>
<td></td>
<td>Indeo Int.</td>
<td>4.95</td>
</tr>
<tr>
<td></td>
<td>Cinepak</td>
<td>18</td>
</tr>
</tbody>
</table>

Table 6.1. Compression Times and File Sizes (AVI),
(Williams et. al. 1996a, p. 28)

Table 6.1 shows how the Cinepak CODEC keeps a relatively constant size for the file independently from the quality chosen. For a 100% quality Indeo Interactive performs better. Microsoft Video 1 is at a disadvantage since it cannot work with more than 16-bit colour.

<table>
<thead>
<tr>
<th>CODEC</th>
<th>Size (MB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>MS Video 1</td>
<td>62.8</td>
</tr>
<tr>
<td>Indeo 3.2</td>
<td>12.6</td>
</tr>
<tr>
<td>Cinepak</td>
<td>18.2</td>
</tr>
</tbody>
</table>

Table 6.2. Compression Times and File Sizes,
Williams et. al. (1996a, p. 29).

The results in table 6.2 were taken for various CODECS using the highest quality, a key frame every frame, and the Quicktime format. This table shows how Indeo 3.2 behaves
better and the size of Cinepak is still about the same as it was when using the AVI format. Contrasting table 1 with table 2, it seems that the Quicktime format is more promising in terms of size than AVI. For example the file sizes using the Indeo 3.2 and Cinepak CODECS is lower. Nevertheless, AVI using the Indeo Interactive CODEC, which is not supported by Quicktime, still performs better than Quicktime.

<table>
<thead>
<tr>
<th>Frame size</th>
<th>CODEC</th>
<th>file size (bytes)</th>
</tr>
</thead>
<tbody>
<tr>
<td>160x120</td>
<td>Indeo 3.2</td>
<td>4420298</td>
</tr>
<tr>
<td></td>
<td>Indeo Interactive</td>
<td>4420814</td>
</tr>
<tr>
<td>192x144</td>
<td>Indeo 3.2</td>
<td>6181010</td>
</tr>
<tr>
<td></td>
<td>Indeo Interactive</td>
<td>6516170</td>
</tr>
<tr>
<td>288x216</td>
<td>Indeo 3.2</td>
<td>11681090</td>
</tr>
<tr>
<td></td>
<td>Indeo Interactive</td>
<td>11399920</td>
</tr>
<tr>
<td>320x240</td>
<td>Indeo 3.2</td>
<td>13280700</td>
</tr>
<tr>
<td></td>
<td>Indeo Interactive</td>
<td>12658408</td>
</tr>
</tbody>
</table>

Table 6.3. File Size vs. Frame Size,
(Williams et al., 1996a, p. 29).

The results shown in table 6.3 (using AVI) suggest that Indeo Interactive becomes better than Indeo 3.2 as the frame size grows, since the file size gets lower.

<table>
<thead>
<tr>
<th>Frame rate</th>
<th>CODEC</th>
<th>File size (bytes)</th>
</tr>
</thead>
<tbody>
<tr>
<td>25</td>
<td>Indeo 3.2</td>
<td>13280700</td>
</tr>
<tr>
<td></td>
<td>Indeo Interactive</td>
<td>12658408</td>
</tr>
<tr>
<td>15</td>
<td>Indeo 3.2</td>
<td>7782486</td>
</tr>
<tr>
<td></td>
<td>Indeo Interactive</td>
<td>7440252</td>
</tr>
<tr>
<td>12.5</td>
<td>Indeo 3.2</td>
<td>6597990</td>
</tr>
<tr>
<td></td>
<td>Indeo Interactive</td>
<td>6331812</td>
</tr>
</tbody>
</table>

Table 6.4. File Size vs. Frame Rate,
(Williams et al., 1996a, p. 30).
The results in table 6.4 were taken at highest quality, key frame every frame, 320x240 frame size and using AVI. They suggest again that currently Indeo Interactive is the most promising CODEC. It shows that Indeo Interactive performs better for any frame rate, since the file sizes are lower.

<table>
<thead>
<tr>
<th>Frame size</th>
<th>initial CODEC</th>
<th>File size (bytes)</th>
<th>MPEG comp. settings</th>
</tr>
</thead>
<tbody>
<tr>
<td>192x144</td>
<td>Intel Raw</td>
<td>611884</td>
<td>high comp</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1713813</td>
<td>high quality comp</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2474927</td>
<td>fast comp</td>
</tr>
<tr>
<td>320x240</td>
<td>Intel Raw</td>
<td>1223695</td>
<td>high comp</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3250012</td>
<td>high quality comp</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4823189</td>
<td>fast comp</td>
</tr>
<tr>
<td>Recompressed using files from table 1</td>
<td>Cinepak</td>
<td>4370476</td>
<td>high quality comp</td>
</tr>
<tr>
<td></td>
<td>MS Video</td>
<td>4245941</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Indeo Interactive</td>
<td>3812733</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>4073239</td>
<td></td>
</tr>
</tbody>
</table>

Table 6.5. MPEG Format,
(Williams et. al, 1996a, p. 30).

Table 6.5 (using the same input) shows that the file sizes using the MPEG format are much smaller, specially when directly compressed to that format. The size is also smaller when the file has been re-compressed from another CODEC. MPEG gives the best quality and size results. It seems to be the most promising in the longer term, (see section 2.3.2.7).

6.2. Performance of the System

This section discusses the results obtained from the evaluation of the performance of PC VEKATS. The results on table 6.6 were taken using two computers, one with a STB Velocity graphic card and the other one with a Matrox Millennium graphic card. Both computers were working at a resolution of 1024x768 pixels with 16-bit colours and
using an Intels' Pentium based computer with 32MB of RAM running at 166 Mhz. Additionally two different versions of OpenGL were tested, one from Microsoft (1996) and the other one from Silicon Graphics (1997).

For the tests different experiments were prepared (see table 6.6). *Situation 1* shows only the *Arthroscopic View*. *Situation 2* shows the *Arthroscopic View* and the *Control Panel*. *Situation 3* shows the *Arthroscopic View*, the *Control Panel* and the *Anatomic Overview*. *Situation 4* shows the *Control Panel* and the *Arthroscopic Overview* but takes out the rigid objects of the knee anatomy. *Situation 5* shows the *Control Panel* and the *Arthroscopic Overview* but takes out the soft objects of the knee anatomy. *Situation 6* shows the *Arthroscopic View*, the *Control Panel* and plays one video clip. *Situation 7* is the same as *Situation 7*, but adds the *Instrument Panel*. *Situation 9* shows the *Arthroscopic View* and the *Instrument Panel* in a collision situation. *Situation 10* shows the *Arthroscopic View* and the *Instrument Panel* in a non-collision situation. In situations 1 to 9, PC VEKATS is detecting a collision of the arthroscope with the knee anatomy. The results obtained from the experiments are discussed in the next paragraphs.
<table>
<thead>
<tr>
<th>Situation</th>
<th>STB Velocity (fps)</th>
<th>Matrox Millennium (fps)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Microsoft's OpenGL</td>
<td>Silicon Graphics' OpenGL</td>
</tr>
<tr>
<td>(1) Arthrosopic View (AV)</td>
<td>0.620604</td>
<td>0.515532</td>
</tr>
<tr>
<td>(2) AV + Control panel (CP)</td>
<td>0.632911</td>
<td>0.536547</td>
</tr>
<tr>
<td>(3) AV + CP + Anatomic overview (AO)</td>
<td>0.516796</td>
<td>0.470640</td>
</tr>
<tr>
<td>(4) CP + (AV - rigid objects)</td>
<td>2.375297</td>
<td>1.691094</td>
</tr>
<tr>
<td>(5) CP + (AV - soft objects)</td>
<td>0.663717</td>
<td>0.646140</td>
</tr>
<tr>
<td>(6) AV + CP + 1 Video clip (VC)</td>
<td>0.502008</td>
<td>0.401667</td>
</tr>
<tr>
<td>(7) AV + VC + AO + CP</td>
<td>0.472408</td>
<td>0.328562</td>
</tr>
<tr>
<td>(8) AV + VC + AO + CP + Instrument panel (IP)</td>
<td>0.484874</td>
<td>0.324505</td>
</tr>
<tr>
<td>(9) AV + In collision + IP</td>
<td>0.528262</td>
<td>0.462046</td>
</tr>
<tr>
<td>(10) AV + No collision + IP</td>
<td>1.101322</td>
<td>2.053388</td>
</tr>
</tbody>
</table>

Table 6.6 PC VEKATS' Performance

The results in table 6.6 show a slight difference between the two cards and the two versions of OpenGL. When both computers were using Microsoft's OpenGL (MOGL), the computer with the Matrox Millenium card (MMC) seemed to perform better than the one with the STB Velocity card (STBC). However, this fact was no longer true when the load of work required from each computer got higher, for example in situations 7 and 8. In fact the MMC seemed to be more affected by the load (and specially by the playing of a video) while the STBC's performance remained more regular and appeared to be better with more load.

The relationship between the results (see table 6.6) when the Silicon Graphic's OpenGL (SGOGL) was used in both computers was not significantly different (except for situations 2, 3 and 4, where the results are better in the STBC).

Table 6.6 also shows that in both computers, in general, the performance using MOGL was better than the performance using SGOGL. This was true except for situation 10,
which was the less loaded situation tested. Here, the performance of SGOGGL doubled that of MOGL.

Some interesting facts are also shown in table 6.6. For example the performance of the system was most affected by the display of the rigid objects (see situations 4 and 5). Besides this, collisions with the anatomy of the knee strongly affected the performance of PC VEKATS (see situations 9 and 10). Finally, when a video was being played (see situations 6, 7, 8) the performance was also highly affected (specially in the MMC).

The relationship between the results shown in table 6.6 may be confusing. Figure 6.1 visualises better the performance of the different configurations when running PC VEKATS together with their relationship.

![Figure 6.1. PC VEKATS' performance](image-url)
Figure 6.1 shows how the difference in the performance between the two versions of OpenGL was more significant than that between the two graphic cards. In fact, the four lines are divided into two groups of two. The higher group (yellow and black lines) represents the performance of the two computers when they were using MOGL. The lower group (red and blue lines) represent the performance of the two computers when they were using SGOGL. It is shown that the difference (in each group) between the performance of the two graphic cards was not really that large. In fact, it could be considered nearly the same. This was true except when a video was being played back. In this case the performance of the MMC was lower. Thus, it is suggested that the MMC performed worse in two dimensional work than the STBC.

It is also proposed from figure 6.1 that SGOGL performed much better than MOGL only when the program (PC VEKATS) was displaying objects using OpenGL commands. Facts like the detection of a collision (non-graphic tasks) strongly affected the performance of SGOCL, which was no longer capable of dealing with the load of the system. In that sense, it is stated that the performance of the MOGL was more regular in the different situations.

6.3. Conclusion

The tables in section 6.1 show the relationship between quality level, compression and file size for various CODECS. The video material (content and quality) and its purpose affect the balance of trade-offs. However, Williams et. al. (1996a) suggests that the assessment of the various CODECS is a subjective process. Thus, the results shown should be considered while viewing the video clips in Williams et. al. (1996b), which are the result of the experiments made.

The results in section 6.2 show that the performance of PC VEKATS is still far from being ideal (about 20 fps, as suggested by the projects' supervisor). Different situations affected in different ways the performance of the system (e.g. video clips, collisions, etc). In general the combination STBC and MOGL behaved better and more regularly in all the situations evaluated. However, if the task dealt only with the display of 3D objects the combination MMC and SGOGL far outperformed the former one. Unfortunately,
this is not the case. In a simulation like PC VEKATS the system must deal not only with the display of the objects, but also with the use of the complex algorithms that build up their behaviour and the extra programs that the system may be running at the same time (e.g. video player).
Chapter 7: Conclusions and Further Work

The complexity of the human knee's joint makes it hard to operate when complaints appear. The currently accepted way to make a diagnostic or a surgical procedure is by performing a form of Minimal Access Surgery called arthroscopy, which minimises the disruption to the surrounding tissue. However, this kind of surgery requires that the surgeon gains a set of skills (e.g. triangulation, orientation, dexterity), since special instruments (e.g. arthroscope, probe, etc) have to be used. As there is no standard training method that helps to develop those skills, a solution could be to implement a Virtual Environments Arthroscopy Training System (VEKATS). Such system, in the long term, could provide a cheap environment where all the skills required can be learned and assessed. The development of a major system like VEKATS is a complex task. In order to achieve this, many techniques have to be applied (e.g. techniques for representing objects, techniques for detecting collisions between the surgical instruments and the anatomy of the knee, etc). The importance of using this techniques is that VEKATS is a computer simulation which requires accurate and realistic behaviour and not just behaviour that looks nice.

Due to the popularity of PCs, VEKATS was re-implemented on this environment. In this way, the new version, PC VEKATS, constitutes a starting point of VEKATS on PC, allowing the evaluation of the system for this platform and, in the future, its expansion to a powerful low cost training system. The object oriented structure of PC VEKATS will allow easier extension or modification of the system. The current source code is almost completely portable across different environments (e.g. UNIX, PC), only a few sections of the implementation (e.g. interface, timing, video playback) are platform dependent. These sections can be changed without altering the heart of the system.

Video clips and the ability to interact with them aid in promoting taught on PC VEKATS. Contrasting with the original implementation of VEKATS, the current video playback system gives the user the possibility to fully interact with the video or even to choose from a video database that can be used to keep different pathologies. Implemented, the AVI video format provides the best playback system standard for a
platform like Windows 95. However, alternatives like MPEG should be further considered in the future.

The current PC VEKATS' interface is more suitable for a training system than that implemented in the original VEKATS (where many of the actions were not self-evident to the user). Many guidelines were applied in its design (e.g. Grouping, Feedback, Consistency, Error Manipulation) and the system no longer relies on a simple configuration file. However, this interface is still a prototype that should be tested in an educational environment. The PC VEKATS' interface is in a state suitable for expansion. For example, the magnetic interaction device is still to be connected, the distortion of the Arthoscop ic View is still to be introduced, etc. This interface will also allow the natural extensions of the system to include new pathologies, increased detail or even other body joints.

The implementation of a PC based VEKATS, with the re-design of the user interface and the video playback system is valuable because it constitutes an improved implementation of VEKATS and the proof that a low-cost system can be implemented for a PC. However, the performance of the PC environment has proven to be poor. Alternatives like parallelisation (e.g. Windows' dynamic data exchange, Winsock) and hardware acceleration (e.g. faster disks, graphic cards, memory, etc) or software acceleration (e.g. better programming tools, better simulation techniques, etc) must be considered in the future.

During this project the author has been involved in many tasks that have helped him to gain a good understanding in many different areas. For example, familiarisation with the tools used, such as, Visual C++, OpenGL, GLUT, Microsoft Foundation Classes. Besides this, the theoretical elements needed to create a major system like PC VEKATS were applied (e.g. representation of objects, collision detection techniques) and the relationship between them understood. General understanding about video playback systems was also achieved. There was, also, an opportunity to see some of the interaction technology that is currently in use in the Virtual Environments laboratory at the University of Hull. Basic knowledge about the knee arthrosco pic procedure was gained too. Additionally, the problems and main issues related with creating a friendly
user-computer interface were found. Finally, techniques such as the Booch's object-oriented analysis and design method were reinforced. It is now clear for the author the complexity that a system like PC VEKATS can reach.
Chapter 8: References


Appendix 1: Source Code

This appendix explains the location (in the enclosed disks) of the programs used to implement PC VEKATS (see chapter 5). Besides this, the file names that implement the most important classes of the system are given.

Disk 1 stores source code of the following programs: the original VEKATS code, an example that shows how to combine the Microsoft Foundation Classes (MFC) with OpenGL, a program that displays a rigid Object in a Canvas, a program that displays a Deformable Object in a Deformable Canvas and a program that uses the MFC, OpenGL and permits interaction.

Disk 2 stores the PC VEKATS’ source code.

Disk 3 stores the source code of the following programs: a program that uses the timing capabilities of Windows 95 and a 16-bits based program that plays a video clip using the Multimedia Control Interface (MCI)’s Set of Functions.

Disk 4 stores the source code of a 32-bit based program that plays a video clip allowing full interaction and using a MCI Window.

The file that keeps the default configuration of PC VEKATS is vekats.cfg. While running, the system will store its performance in the file performa.txt. Note that this file is not cleared each time PC VEKATS is run. The information is just appended. The file Mf cog11.mdp is the project’s file.

The classes of PC VEKATS that build up the interface were implemented as follows: the Application class was implemented in the Mfcog1.h and Mfcog1.cpp files. The View class was implemented in the Mfcoglww.h and Mfcoglww.cpp files. The Frame window class was implemented in the Mainfrm.h and Mainfrm.cpp files. The Document class was implemented in the Mfcgldoc.h and Mfcgldoc.cpp files. The Control Panel class
was implemented in the Control.h and Control.cpp files. Finally, the Instrument Panel class was implemented in the Instrument.h and Instrument.cpp files.

The classes of PC VEKATS that build up the simulation were implemented as follows: the Simulation class was implemented in the Mainsim.h and Mainsim.cpp files. The Object class was implemented in the Objects.h and Objects.cpp files. The Deformable Object class was implemented in the Defoobject.h and Defoobject.cpp files. The Spring-and-Dashpot, Reference Frame and Finite Element Modelling classes were implemented in the files Spring, Refframe and FEM (.h and .cpp) files respectively. The Canvas, Deformable Canvas and Arthrocanvas classes were implemented in the Canvas, Defcanvas and Arthrocanvas (.h and .cpp) files respectively. The Matrix 1D and Matrix 2D classes were implemented in the Matrix.h and Matrix.cpp files. Finally, The Vector class was implemented in the Vector.h file. The rest of the classes shown in section 4.1 can be easily found in the files named in this paragraph or in other files which names are the same as those of the classes that they implement.